TUPaD Programación I
Alumnos: Castro, Claudio David Comisión: 6

Castro, Ramón Gabriel

Análisis de Algoritmos

Comparación de Logaritmos usando análisis Empírico y Teórico

Introducción:

En el ámbito de la informática, los algoritmos son la esencia de la resolución de problemas computacionales. Un algoritmo puede definirse como una secuencia lógica y finita de pasos diseñados para realizar una tarea específica, cumpliendo con requisitos fundamentales como obtener el resultado esperado, el manejo adecuado de entradas inesperadas y el uso óptimo de recursos. Dado que la programación no solo se trata de escribir código funcional, sino también de garantizar que las soluciones sean escalables y eficientes, el análisis de algoritmos se convierte en una disciplina esencial para cualquier técnico o profesional en el área.

La elección de este tema radica en su importancia práctica: en el desarrollo de software, no basta con que un programa funcione; también debe hacerlo de manera óptima, especialmente cuando se trabaja con grandes volúmenes de datos o sistemas de alto rendimiento. Comprender cómo evaluar y comparar algoritmos permite tomar decisiones informadas, ya sea al seleccionar una estructura de datos, optimizar un proceso existente o diseñar nuevas soluciones.

Este trabajo tiene como objetivo principal explorar los fundamentos del análisis de algoritmos, abordando tanto aspectos teóricos como empíricos, para establecer criterios que permitan determinar su eficiencia en distintos contextos. Además, se busca destacar su relevancia en la formación de un técnico en programación, ya que estas competencias son clave para desarrollar software de calidad, adaptable a diferentes necesidades y restricciones.

Marco Teórico:

El análisis de algoritmos constituye un pilar fundamental en la programación, permitiendo evaluar la eficiencia de las soluciones mediante métricas objetivas. Este estudio se centra en comparar dos algoritmos que realizan el cálculo factorial, para ello usamos definiciones como:

-Enfoque iterativo: Basado en estructuras de control repetitivas.

-Enfoque recursivo: Basado en autollamadas con casos base.

-Factorial: es el producto de todos los números enteros positivos menores hasta llegar a ‘n’

-Función T(n): proporciona una medición exacta del costo computacional

-Big-O: muestra la tendencia del crecimiento de una función T(n)

Objetivo principal: Determinar cuál algoritmo realiza un mejor uso de recursos computacionales bajo diferentes datos de entrada.

Caso Práctico:

Para hacer el análisis de algoritmos que calculan el numero factorial, implementamos dos algoritmos, uno calcula la factorial de forma recursiva y el otro de forma iterativa.

- **Análisis Empírico: Implementación de los algoritmos:**

```
import time

# Version Recursiva

def factorial_rec(num):

  if num==0:

    return 1

  else:

    return num * factorial_rec(num-1)

# Medición del tiempo de ejecucion

print("\t ---*** Versión Recursiva ***---")

print("n\t factorial de n Tiempo ejecucion(ms)")

for i in range(0,51,5):

  inicio = time.perf_counter()

  resultado = factorial_rec(i)
```

```
  fin = time.perf_counter()
  tiempo_ejecucion_rec = (fin - inicio) * 1000  # Tiempo en milisegundos
  print(f"{i}\t {resultado:.4e}\t {tiempo_ejecucion_rec:.4e} ")
print("----------------------------------------")
# Version Iterativa
def factorial_iter(n):
  fact = 1
  for i in range(1,n+1):
    fact *= i
  return fact
print("\t ---*** Metodo Iterativo ***---")
print("n\t factorial de n  Tiempo ejecucion(ms)")
for i in range(0,51,5):
  inicio = time.perf_counter()
  resultado = factorial_iter(i)
  fin = time.perf_counter()
  tiempo_ejecucion_iter = (fin - inicio) * 1000  # Tiempo en milisegundos
  print(f"{i}\t {resultado:.4e}\t {tiempo_ejecucion_iter:.4e} ")
```

- **Análisis Teórico:**

```
# Version Iterativa

def factorial_iter(n):

  fact = 1                          # 1 operacion

  for i in range(1,n+1):            # n

    fact *= i                       # 2 operaciones

  return fact                       # 1 operación
```

Calculo de T(n): 1 + 1 + 2*n = 2*n + 2

```
# Version Recursiva

def factorial_rec(num):

  if num==0:                                # 1 operacion

    return 1

  else:

    return num * factorial_rec(num-1)       # 1+T(n-1) operacines
```

Calculo de T(n): tiene dos valores, uno para el caso base y el otro para cuando n es mayor que cero → 1 cuando n=0 ; 1+T(n-1) cuando n>0

Aplicamos la técnica de despliegue de recurrencia:

T1(n) = 1+ T(n-1)

T2(n) = 1+(1+ T(n-1)) = 2 + T(n-2)

T3(n) = 1+(2+T(n-2)) = 3 + T(n-3)

.. .. …

Ti(n) = i + T(n-i) i es el número total de llamadas recursivas

Como llegamos al caso base entonces: (n-i) = 0 → n = i , con lo que obtenemos

T(n) = n + 1

-Obtenemos las dos T(n): TA(n) = 2n+2 ; TB(n) = n+1

Para poder obtener el Big-O necesitamos realizar dos pasos:

1- Obtenemos el termino de mayor crecimiento de cada función:

TA(n) = 2n

TB(n) = n

2- Eliminamos el número que multiplique a la variable.

TA(n) = n

TB(n) = n

Como resultado obtenemos que las cotas superiores(Big-O) son las mismas

Metodología Utilizada:

La elaboración del trabajo se realizó en las siguientes etapas:

- Recolección de información teórica en documentación disponible.
- Implementación en Python de los algoritmos estudiados.
- Pruebas con diferentes conjuntos de datos.
- Registro de resultados y validación de funcionalidad.
- Elaboración del informe y preparación de anexos.

Resultados Obtenidos:

-los programas corren correctamente.

-se pudo obtener los tiempos de ejecución de cada algoritmo.

-se pudo encontrar la función matemática T(n) de cada algoritmo.

-pudo hacerse las comparaciones de rendimiento entre los algoritmos.

Conclusiones:

El análisis comparativo, empírico, entre los algoritmos iterativo y recursivo para el cálculo factorial demostró que, aunque su desempeño práctico difiere en el momento de usar tiempo de ejecución. El algoritmo iterativo mostró mayor eficiencia en tiempos de ejecución. También este método empírico permite obtener graficas de tiempo de ejecución, para poder hacer una comparación visual de los datos obtenidos.

Con respecto al análisis teórico podemos decir que identifica la complejidad computacional de los algoritmos; ambos algoritmos tienen la misma cota superior o Big-O, por lo tanto, según este análisis ambos códigos tienen un comportamiento similar con lo cual no se puede afirmar que un algoritmo sea mejor que el otro.

Además, el trabajo destacó la complementariedad entre el análisis empírico (que proporciona datos concretos de rendimiento) y el análisis teórico (que ofrece una generalización abstracta).

Bibliografía

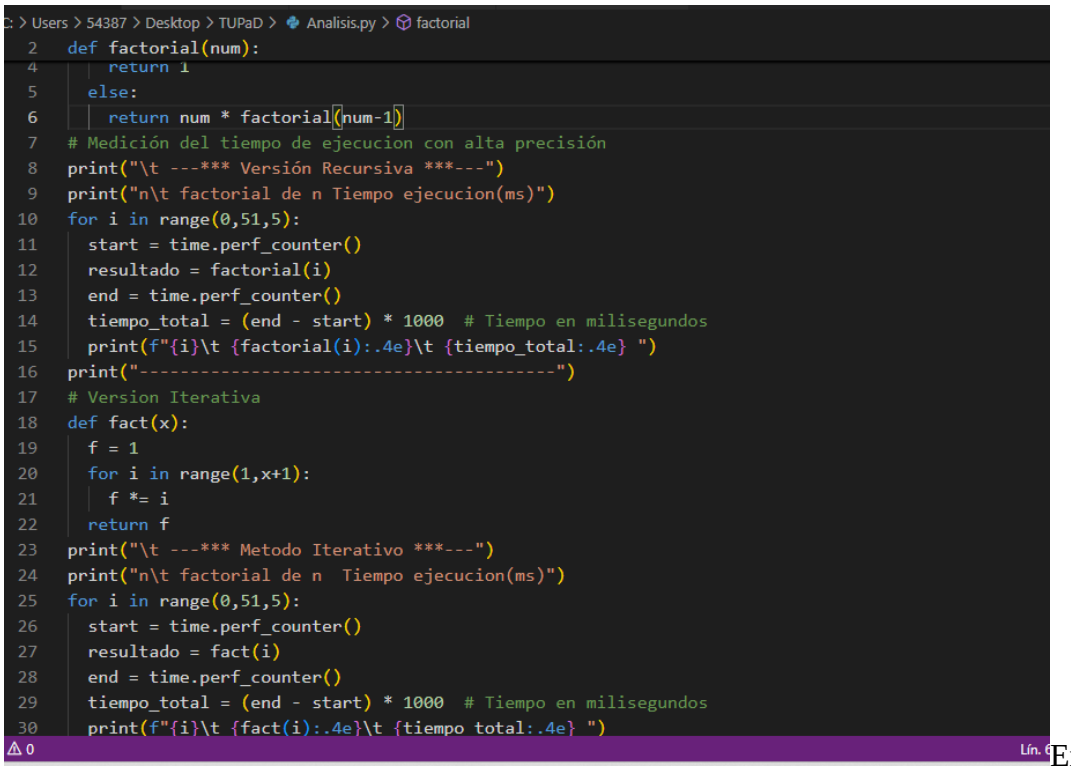
Facultad de Ingeniería, Universidad de la República. (s. f.). Estructura de datos y algoritmos: Introducción al análisis de algoritmos. https://www.fing.edu.uy/tecnoinf/mvd/cursos/eda/material/teo/EDA-teorico3.pdf

- Duch, A. (2007, marzo). Análisis de algoritmos. Barcelona. https://www.cs.upc.edu/~duch/home/duch/analisis.pdf
- Análisis de algoritmos recursivos de Jorge Gozález Mollá en canal de youtube: https://www.youtube.com/watch?v=1Bna7tbjIDI&t=601s
- Universidad Tecnológica Nacional. (s. f.). Material de lectura de la cátedra Programación I: Tecnicatura Universitaria en Programación.

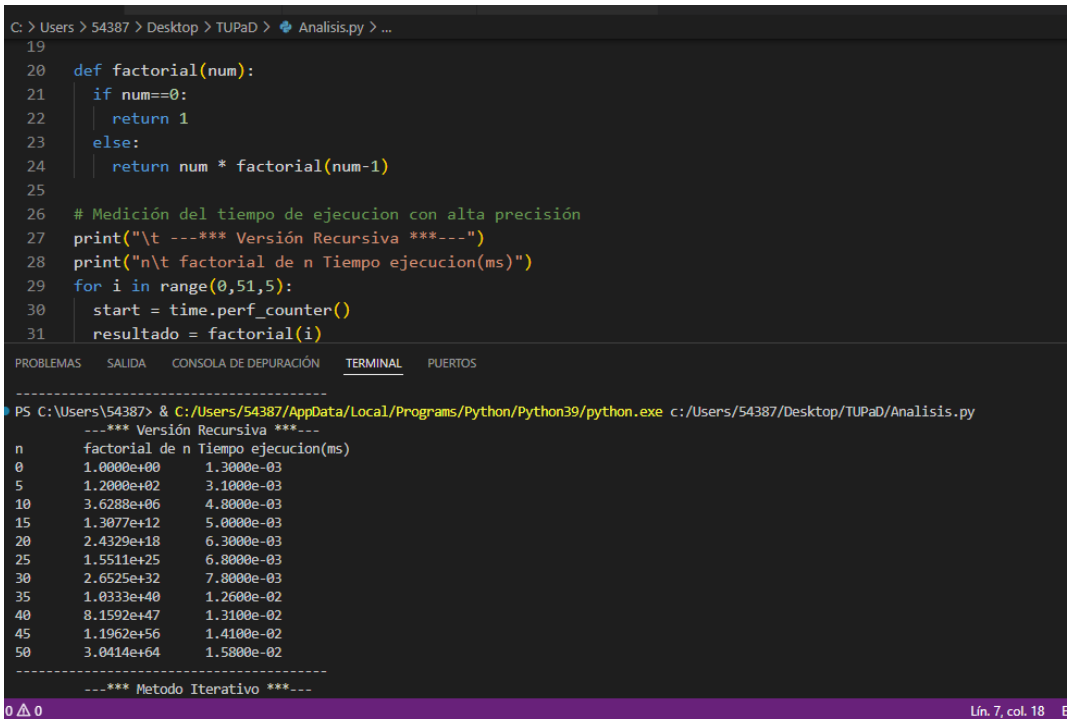
Anexos

- Captura de pantalla del programa funcionando.

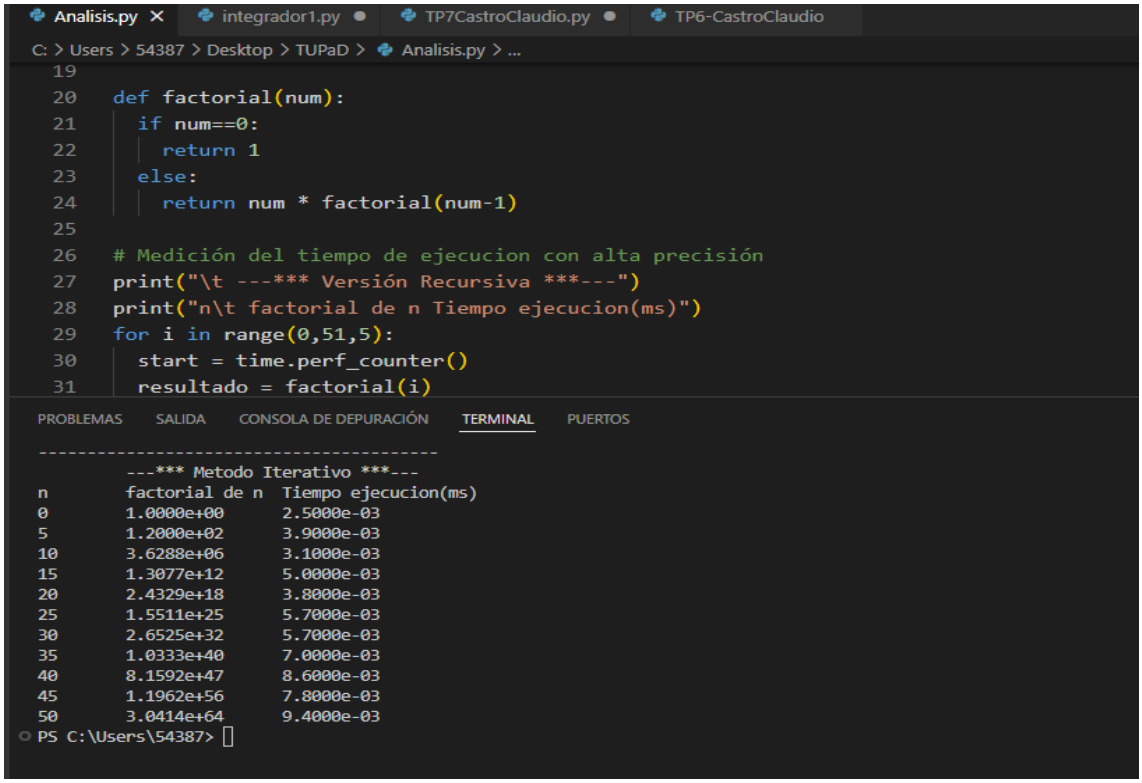
En la siguiente foto se observa el código de los dos algoritmos en visual studio code.



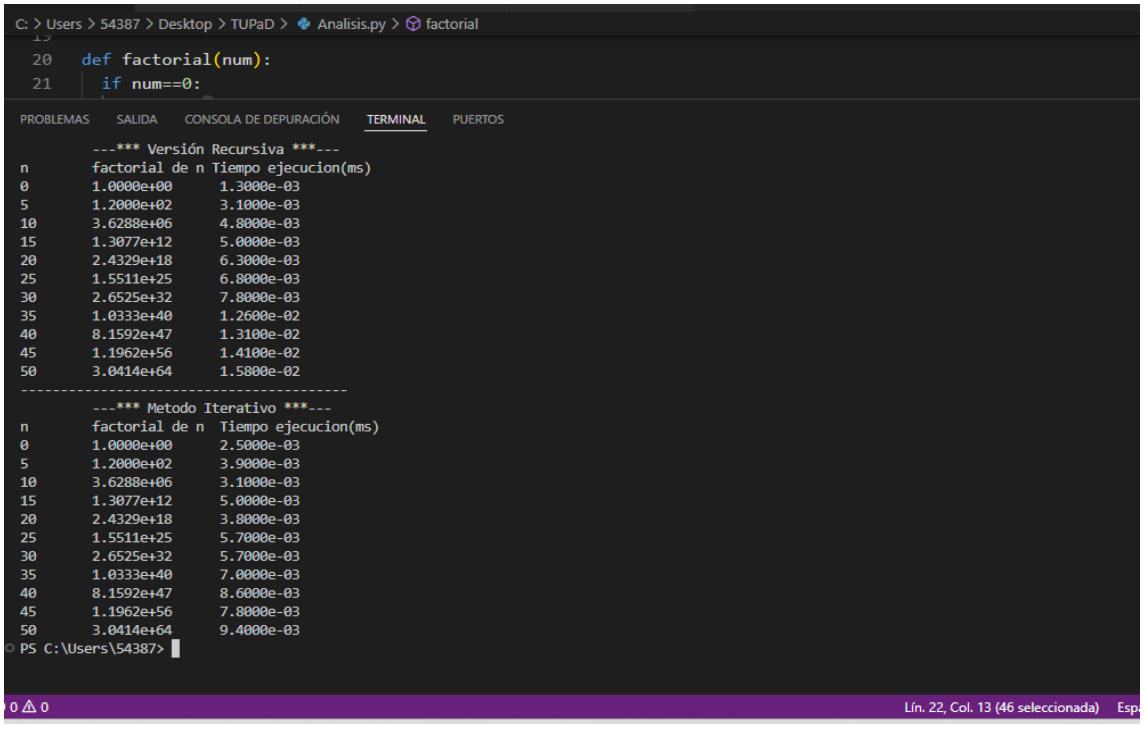
En la siguiente foto se observa cuando se hace correr el programa y muestra como resultado la versión Recursiva.



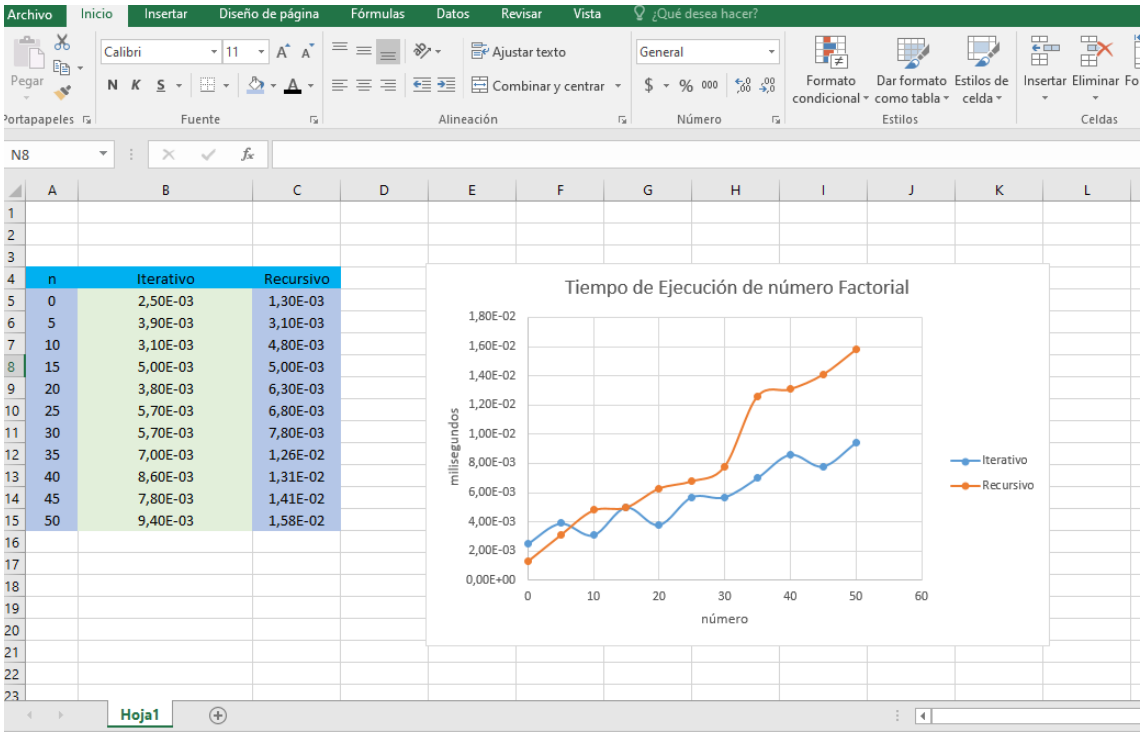
En la siguiente foto se observa cuando se hace correr el programa y muestra como resultado la versión Iterativa.



En la siguiente foto se observa la salida de ambos algoritmos.



En la siguiente foto se observa la gráfica de los datos obtenidos.



n	Iterativo	Recursivo
0	2,50E-03	1,30E-03
5	3,90E-03	3,10E-03
10	3,10E-03	4,80E-03
15	5,00E-03	5,00E-03
20	3,80E-03	6,30E-03
25	5,70E-03	6,80E-03
30	5,70E-03	7,80E-03
35	7,00E-03	1,26E-02
40	8,60E-03	1,31E-02
45	7,80E-03	1,41E-02
50	9,40E-03	1,58E-02